OBDDs and (Almost) k-wise Independent Random Variables

Marc Bury*
TU Dortmund, LS2 Informatik, Germany

OBDD-based graph algorithms deal with the characteristic function of the edge set E of a graph $G = (V, E)$ which is represented by an OBDD and solve optimization problems by mainly using functional operations. We present an OBDD-based algorithm which uses randomization for the first time. In particular, we give a maximal matching algorithm with $O(\log^3 |V|)$ functional operations in expectation. This algorithm may be of independent interest. The experimental evaluation shows that this algorithm outperforms known OBDD-based algorithms for the maximal matching problem.

In order to use randomization, we investigate the OBDD complexity of 2^n (almost) k-wise independent binary random variables. We give a OBDD construction of size $O(n)$ for 3-wise independent random variables and show a lower bound of $2^{\Omega(n)}$ on the OBDD size for $k \geq 4$. The best known lower bound was $\Omega(2^n/n)$ for $k \approx \log n$ due to Kabanets [24]. We also give a very simple construction of 2^n (ε, k)-wise independent binary random variables by constructing a random OBDD of width $O(nk^2/\varepsilon)$.

*Supported by Deutsche Forschungsgemeinschaft, grant BO 2755/1-2.

1 Introduction

In times of Big Data, classical algorithms for optimization problems quickly exceed feasible running times or memory requirements. For instance, the rapid growth of the Internet and social networks results in massive graphs which traditional algorithms cannot process in reasonable time or space. In order to deal with such graphs, implicit (symbolic) algorithms have been investigated where the input graph is represented by the characteristic function χ_E of the edge set and the nodes are encoded by binary numbers. Using *Ordered Binary Decision Diagrams* (OBDDs), which were introduced by Bryant [12], to represent χ_E can significantly decrease the space needed to store such graphs. Furthermore, using mainly functional operations, e. g., binary synthesis and quantifications, which are efficiently supported by the OBDD data structure, many optimization problems can be solved on OBDD represented inputs ([17, 18, 20, 35, 36, 37, 42]). Implicit algorithms were successfully applied in many areas, e. g., model checking [13], integer linear programming [25] and logic minimization [15]. With one of the first implicit graph algorithms, Hachtel and Somenzi [20] were able to compute a maximum flow on 0-1-networks with up to 10^{36} edges and 10^{27} nodes in reasonable time.

There are two main parameters influencing the actual running time of OBDD-based algorithms: the number of functional operations and the sizes of all intermediate OBDDs used during the computation. The size of OBDDs representing graphs was investigated for bipartite graphs [33], interval graphs [33, 19], cographs [33] and graphs with bounded tree- and clique-width [29]. Bounding the sizes of the other OBDDs, which can occur during the computation, is quite difficult and could only be proven for very structured input graphs like grid graphs [8, 42]. In terms of functional operations, Sawitzki [38] showed that the set of problems solved by an implicit algorithm using $O(\log^k N)$ functional operations and functions defined on $O(\log N)$ variables is equal to the complexity class FNC, i. e., the class of all optimization problems that can be efficiently solved in parallel. Implicit algorithms with these properties were designed for instance for topological sorting [42], minimum spanning tree [6], metric TSP approximation [7] and maximal matching [10] where a matching M, i. e., a set of edges without a common vertex, is called maximal if M is no proper subset of another matching. However, Sawitzki's structural result yields neither a good transformation of parallel algorithms to implicit algorithms nor does it give a statement about the actual performance of the implicit algorithms. Nevertheless, designing implicit algorithms for optimization problems is not only an adaption of parallel algorithms but can give new insights into the problems. For example, Gentilini et al. [18] introduced a new notion of spine-sets in the context of implicit algorithms for connectivity related problems. When analyzing implicit algorithms, the actual running time can either be proven for very structured input graphs like [42] did for topological sorting and [8] for maximum matching or the running time is experimentally evaluated like in [20] for maximum flows and in [8, 19] for maximum matching on bipartite graphs or unit interval graphs.

Overall there seems to be a trade-off: The number of operations is an important measure of difficulty [5] but decreasing the number of operations often results in an increase of the number of variables of the used functions. Since the worst case OBDD size

of a function $f : \{0,1\}^n \to \{0,1\}$ is $\Theta(2^n/n)$, the number of variables should be as small as possible to decrease the worst case running time. This trade-off was also empirically observed. For instance, an implicit algorithm computing the transitive closure that uses an iterative squaring approach and a polylogarithmic number of operations is often inferior to an implicit sequential algorithm, which needs a linear number of operations in worst case [5, 21]. Another example is the maximal matching algorithm (BP) of Bollig and Pröger [10] that uses only $O(\log^4 N)$ functional operations on functions with at most $6 \log N$ variables while the algorithm (HS) of Hachtel and Somenzi [20] uses $O(N \log N)$ operations in the worst case on function with at most $3 \log N$ variables. However, HS is clearly superior to BP on most instances (see Section 5). An additional reason might be its simplicity.

Using randomization in an explicit algorithm often leads to simple and fast algorithms. Here, we propose the first attempt at using randomization to obtain algorithms which have both a small number of variables and a small expected number of functional operations. For this, we want to represent random functions $f_r : \{0,1\}^n \to \{0,1\}$ with $\Pr[f_r(x) = 1] = p$ for every $x \in \{0,1\}^n$ and some fixed probability $0 < p < 1$ by OBDDs. Using random functions in implicit algorithms is difficult. We need to construct them efficiently but, obviously, if the function values are completely independent (and p is a constant), then the OBDD (and even the more general FBDD or read-once branching program) size of f_r is exponentially large with an overwhelming probability [40]. Thus, we investigate the OBDD size and construction of (almost) k-wise independent random functions where the distribution induced on every k different function values is (almost) uniform.

Related Work

A succinct representation of 2^n random bits, which are k-wise independent, was presented by Alon et al. [1] using $\lfloor k/2 \rfloor n + 1$ independent random bits. This number of random bits is very close to the lower bound of Chor and Goldreich [14]. In order to reduce the number of random bits even further, Naor and Naor [31] introduced the notion of almost k-wise independence where the distribution on every k random bits is "close" to uniform. Constructions of almost k-wise independent random variables are also given in [2] and are using only at most $2(\log n + \log k + \log(1/\varepsilon))$ random bits where ε is a bound on the closeness to the uniform distribution. Looking for a simple representation of almost k-wise independent random variables, Savický [34] presented a Boolean formula of constant depth and polynomial size and used $n \log^2 k \log(1/\varepsilon)$ random bits. In all of these constructions, the running time of computing the i-th random bit with $0 \leq i \leq 2^n - 1$ depends on k and ε.

Such small probability spaces can be used for a succinct representation of a random string of length 2^n, e. g., in streaming algorithms [3], or for derandomization [1, 27]. The randomized parallel algorithms from [1, 27] compute a maximal independent set (MIS) of a graph, i. e., a subset I of V such that no two nodes of I are adjacent and any vertex in G is either in I or is adjacent to a node of I. The computation of a MIS has also been extensively studied in the area of distributed algorithms [4, 26]. An optimal

randomized distributed MIS algorithm was presented in [30] where the time and bit complexity (bits per channel) is $O(\log N)$. Using completely independent random bits, Israeli and Itai [22] give a randomized parallel algorithm computing a maximal matching in time $O(\log N)$.

While we are looking for k-wise independent functions with small OBDD size, Kabanets [24] constructed simple Boolean functions which are hard for FBDDs by investigating (almost) $\Theta(n)$-wise independent random functions and showed that the probability tends to 1 as n grows that the size is $\Omega(2^n/n)$.

Our Contribution.

In Section 3, we show that the OBDD and FBDD size is at least $2^{\Omega(n+\log(p'))}$ with $p' = 2p(1-p)$ if the function values of f_r are k-wise independent with $k \geq 4$. We give an efficient construction of OBDDs for 3-wise independent random functions which is based on the known construction of 3-wise independent random variables using BCH-schemes [1]. In Section 4 we investigate a simple construction of a random OBDD due to Bollig and Wegener [11] which generates almost k-wise independent random functions and has size $O((kn)^2/\varepsilon)$. Reading the actual value of the i-th random bit is just an evaluation of the function on input i which can be done in $O(n)$ time, i. e., it is independent of both k and ε. This construction is used as an input distribution for our implicit algorithm in the experimental evaluation. In Section 5 we use pairwise independent random functions to design a simple maximal matching algorithm that uses only $O(\log^3 N)$ functional operations in expectation and functions with at most $3 \log N$ variables. This algorithm can easily be extended to the MIS problem and can be implemented as a parallel algorithm using $O(\log N)$ time in expectation or as a distributed algorithm with $O(\log N)$ expected time and bit complexity (and is simpler than in [30]). To the best of our knowledge, this is the first (explicit or implicit) maximal matching (or independent set) algorithm that does not need any knowledge about the graph (like size or node degrees) as well as uses only pairwise independent random variables. Eventually, we evaluate this algorithm empirically and show that known implicit maximal matching algorithms are outperformed by the new randomized algorithm.

2 Preliminaries

Binary Decision Diagrams

We denote the set of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ by B_n. For $x \in \{0,1\}^n$ denote the value of x by $|x| := \sum_{i=0}^{n-1} x_i \cdot 2^i$. Further, for $l \in \mathbb{N}$, we denote by $[l]_2$ the corresponding binary number of l, i. e., $|[l]_2| = l$. In his seminal paper [12], Bryant introduced *Ordered Binary Decision Diagrams* (OBDDs), that allow a compact representation of not too few Boolean functions and also supports many functional operations efficiently.

Definition 2.1 (Ordered Binary Decision Diagram (OBDD))**.**

Order. A *variable order* π on the input variables $X = \{x_0, \ldots, x_{n-1}\}$ of a Boolean function $f \in B_n$ is a permutation of the index set $I = \{0, \ldots, n-1\}$.

Representation. A π-*OBDD* is a directed, acyclic, and rooted graph G with two sinks labeled by the constants 0 and 1. Each inner node is labeled by an input variable from X and has exactly two outgoing edges labeled by 0 and 1. Each edge (x_i, x_j) has to respect the variable order π, i. e., $\pi(i) < \pi(j)$.

Evaluation. An assignment $a \in \{0,1\}^n$ of the variables defines a path from the root to a sink by leaving each x_i-node via the a_i-edge. A π-OBDD G_f represents f iff for every $a \in \{0,1\}^n$ the defined path ends in the sink with label $f(a)$.

Complexity. The *size* of a π-OBDD G, denoted by $|G|$, is the number of nodes in G. The π-OBDD size of a function f is the minimum size of a π-OBDD representing f. The OBDD size of f is the minimum π-OBDD size over all variable orders π. The *width* of G is the maximum number of nodes labeled by the same input variable.

The more general read-once branching programs or *Free Binary Decision Diagrams* (FBDDs) were introduced by Masek [28]. In an FBDD every variable can only be read once on a path from the root to a sink (but the order is not restricted).

A simple function is the inner product $IP_n(x,y) = \bigoplus_{i=0}^{n-1} x_i \wedge y_i$ of two vectors $x, y \in \{0,1\}^n$. Let π be a variable order where for every $0 \leq i \leq n$, the variables x_i and y_i are consecutive. It is easy to see that the π-OBDD representing IP_n has size $O(n)$ and width 2. Notice that the π-OBDD size is still $O(n)$ if we replace an input vector, e. g., y, by a constant vector $r \in \{0,1\}^n$.

In the following we describe some important operations on Boolean functions which we will use in this paper (see, e. g., Section 3.3 in [41] for a detailed list). Let f and g be Boolean functions in B_n on the variable set $X = \{x_0, \ldots, x_{n-1}\}$, π a fixed order and let G_f and G_g be π-OBDDs representing f and g, respectively. We denote the subfunction of f where x_j for some $0 \leq j \leq n-1$ is replaced by a constant $a \in \{0,1\}$ by $f_{|x_j=a}$.

1. **Negation:** Given G_f, compute a representation for the function $\overline{f} \in B_n$. Time: $O(1)$

2. **Replacement by constant:** Given G_f, an index $i \in \{0, \ldots, n-1\}$, and a Boolean constant $c_i \in \{0,1\}$, compute a representation for the subfunction $f_{|x_i=c_i}$. Time: $O(|G_f|)$

3. **Equality test:** Given G_f and G_g, decide whether f and g are equal. Time: $O(1)$ in most implementations (when using so called *Shared OBDDs*, see [41]), otherwise $O(|G_f| + |G_g|)$.

4. **Synthesis:** Given G_f and G_g and a binary Boolean operation $\otimes \in B_2$, compute a representation for the function $h \in B_n$ defined as $h := f \otimes g$. Time: $O(|G_f| \cdot |G_g|)$

5. **Quantification:** Given G_f, an index $i \in \{1, \ldots, n\}$ and a quantifier $Q \in \{\exists, \forall\}$, compute a representation for the function $h \in B_n$ defined as $h := Qx_i : f$ where

$\exists x_i : f := f_{|x_i=0} \vee f_{|x_i=1}$ and $\forall x_i : f := f_{|x_i=0} \wedge f_{|x_i=1}$. Time: see replacement by constant and synthesis

In addition to the operations mentioned above, in implicit graph algorithms (see the next section) the following operation (see, e. g., [37]) is useful to reverse the edges of a given graph. We will use this operation implicitly by writing for instance $f(x, y)$ and $f(y, x)$ in the pseudo code of our algorithm.

Definition 2.2. Let $k \in \mathbb{N}$, ρ be a permutation of $\{1, \ldots, k\}$ and $f \in B_{kn}$ with input vectors $x^{(1)}, \ldots, x^{(k)} \in \{0, 1\}^n$. The argument reordering $\mathcal{R}_\rho(f) \in B_{kn}$ with respect to ρ is defined by $\mathcal{R}_\rho(f)(x^{(1)}, \ldots, x^{(k)}) := f(x^{(\rho(1))}, \ldots, x^{(\rho(k))})$.

This operation can be computed by just renaming the variables and repairing the variable order using $3(k-1)n$ functional operations (see [9]).

A function f *depends essentially* on a variable x_i iff $f_{|x_i=0} \neq f_{|x_i=1}$. A characterization of minimal π-OBDDs due to Sieling and Wegener [39] can often be used to bound the OBDD size.

Theorem 2.3 ([39]). *Let $f \in B_n$ and for all $i = 0, \ldots, n-1$ let s_i be the number of different subfunctions which result from replacing all variables $x_{\pi(j)}$ with $0 \leq j \leq i-1$ by constants and which essentially depend on $x_{\pi(i)}$. Then the minimal π-OBDD representing f has s_i nodes labeled by $x_{\pi(i)}$.*

Lower bound techniques for FBDDs are similar but have to take into account that the order can change for different paths. The following property due to Jukna [23] can be used to show good lower bounds for the FBDD size.

Definition 2.4. A function $f \in B_n$ with input variables $X = \{x_0, \ldots, x_{n-1}\}$ is called *r-mixed* if for all $V \subseteq X$ with $|V| = r$ the 2^r assignments to the variables in V lead to different subfunctions.

Lemma 2.5 ([23]). *The FBDD size of a r-mixed function is bounded below by $2^r - 1$.*

OBDD-Based Graph Algorithms

Let $G = (V, E)$ be a directed graph with node set $V = \{v_0, \ldots, v_{N-1}\}$ and edge set $E \subseteq V \times V$. Here, an undirected graph is interpreted as a directed symmetric graph. Implicit algorithms work on the characteristic function $\chi_E \in B_{2n}$ of E where $n = \lceil \log N \rceil$ is the number of bits needed to encode a node of V and $\chi_E(x, y) = 1$ if and only if $(v_{|x|}, v_{|y|}) \in E$. Often it is also necessary to store the valid encodings of nodes by the characteristic function χ_V of V. Besides functional operations, OBDD-based algorithms can use $O(\text{polylog}\,|V|)$ additional time, e. g., for constructing OBDDs for a specific function (equality, greater than, inner product, ...).

Small probability spaces

A succinct representation of our random function is essential for our randomized implicit algorithm. For this, we have to used random functions with limited independence.

Definition 2.6 ((Almost) k-wise independence)**.** Let $X_0, \ldots, X_{m-1}$ be m binary random variables. These variables are called *k-wise independent* with $k \leq m$ if and only if for all $0 \leq i_1 < \ldots i_k \leq m-1$ and for all $l_1, \ldots, l_k \in \{0,1\}$

$$\Pr\left[X_{i_1} = l_1 \wedge \ldots \wedge X_{i_k} = l_k\right] = 2^{-k}$$

and they are called *(ε, k)-wise independent* iff

$$\left|\Pr\left[X_{i_1} = l_1 \wedge \ldots \wedge X_{i_k} = l_k\right] - 2^{-k}\right| \leq \varepsilon.$$

The BCH scheme introduced by Alon et. al [1] is a construction of k-wise independent random variables $X_0, \ldots, X_{2^n-1}$ that only needs $\lfloor k/2 \rfloor n + 1$ independent random bits and works as follows: Let $r_n \in \{0,1\}$ be a random bit, $r^{(j)} \in \{0,1\}^n$ for $1 \leq j \leq l$ be l uniformly random row vectors, and let the row vector $r = \left[r^{(1)}, \ldots, r^{(l)}\right] \in \{0,1\}^{ln+1}$ be the concatenation of the vectors. For $0 \leq i \leq 2^n - 1$ define $X_i = IP_{ln+1}\left(r, \left[[i]_2, [i^3]_2, \ldots, \left[i^{2l-1}\right]_2\right]\right) \oplus r_n$ where i^{2j-1} for $j = 1, \ldots, l$ is computed in the finite field $GF(2^n)$. This scheme generates $2l+1$-wise independent random bits [1] (if we exclude X_0 and if r_n is dropped we obtain $2l$-wise independence).

We say a function $f_r : \{0,1\}^n \to \{0,1\}$ is a k-wise ((ε, k)-wise) independent random function iff the random variables $X_i = f_r([i]_2)$ with $0 \leq i \leq 2^n - 1$ are k-wise ((ε, k)-wise) independent. The BCH scheme gives us an intuition of the complexity of an OBDD representing a k-wise independent function: For $k \leq 3$ the random variables of the BCH scheme are $X_i = IP_{ln+1}(r, [i]_2) \oplus r_n$ which is basically a simple inner product of two binary vectors. For $k \geq 4$, i. e., $l \geq 2$, we have to multiply in a finite field to generate the random variables. Since multiplication is hard for OBDDs it seems likely that k-wise independent functions for $k \geq 4$ are also hard.

3 OBDD Size of k-wise Independent Random Functions

We start with some upper bounds on the OBDD size of 3-wise independent random functions. Notice that by means of the BCH scheme it is not possible to construct a pairwise independent function (which is not 3-wise independent) since $X_0 = IP(r, 0^n) = 0$ for every $r \in \{0,1\}^n$.

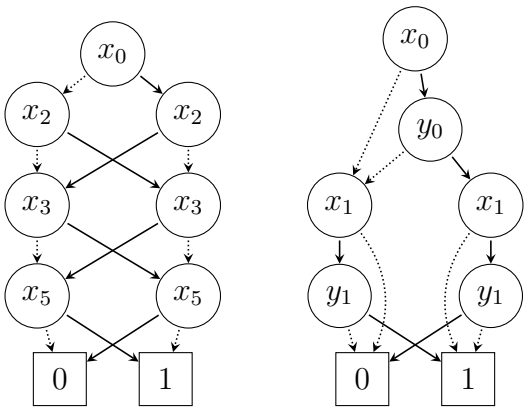

Figure 1: Two π-OBBDs with $\pi = (x_0, y_0, \ldots, x_{n-1}, y_{n-1})$ for the functions $IP_6(x, y)$ where y is replaced by the constant vector $(1, 0, 1, 1, 0, 1)$ and $IP_2(x, y)$.

Theorem 3.1. *Let $\varepsilon > 0$, $n \in \mathbb{N}$, p be a probability with $1/2^n \leq p \leq 1/2$, and π be a variable order on the input variables $\{x_0, \ldots, x_{n-1}\}$. Define $p(x) := \Pr_{r \in \{0,1\}^{n+1}}[f_r(x) = 1]$.*

1. *We can construct an π-OBDD representing a 3-wise independent function $f_r : \{0,1\}^n \to \{0,1\}$ in time $O(n)$ such that $p(x) = 1/2$ for every $x \in \{0,1\}^n$, and the size of the π-OBDD is $O(n)$ with width 2 for every $r \in \{0,1\}^{n+1}$ (see Algorithm 1).*

2. *We can construct an π-OBDD representing a 3-wise independent function $f_r : \{0,1\}^n \to \{0,1\}$ in time $O(\frac{n}{p \cdot \varepsilon})$ such that $\frac{\lceil p \cdot 2^n \rceil}{2^n} \leq p(x) \leq (1+\varepsilon) \cdot \frac{\lceil p \cdot 2^n \rceil}{2^n}$ for every $x \in \{0,1\}^n$, and the size of the π-OBDD is bounded above by $O(\frac{n}{p \cdot \varepsilon})$ for every $r \in \{0,1\}^{n+1}$.*

3. *We can compute a function $g_A(x) : \{0,1\}^n \leftarrow \{0,1\}$ for a random matrix $A \in \{0,1\}^n \times \{0,1\}^n$ such that $\Pr_A[g_A(x) = 1] = \dfrac{\lceil p \cdot 2^n \rceil}{2^n}$ using $O(n)$ functional operations. Furthermore, using also $O(n)$ functional operations we can compute a priority function $GT_A(x, y)$, which is equal to 1 iff $|v_x| > |v_y|$, where $v_z \in \{0,1\}^n$ for all $z \in \{0,1\}^n$ and $(v_0, \ldots, v_{2^n-1})$ is a pairwise independent random permutation of $\{0,1\}^n$. Note: This construction is also possible using only $2n$ random bits by computing single bits of $a_0 + |x| \cdot a_1$ in $\mathbb{F}_{2^n}$.*

Proof. 1. This is an implication of the BCH scheme for 3-wise independent random bits. Recall that for random $r = (r_0, \ldots, r_n) \in \{0,1\}^{n+1}$ the random variables $X_i(r) = IP(r, [1, [i]_2])$ for $0 \leq i \leq 2^n - 1$ are 3-wise independent and $\Pr[X_i = 1] = 1/2$ for every i. We define $f_r(x) = X_{|x|}(r)$ (see Algorithm 1). As described in the preliminaries, the function IP can be represented by an OBDD of width 2 and size $O(n)$ for any variable order if one input vector is replaced by a constant vector. The construction of the OBDD

is straightforward (see, e. g., Fig. 1) and can be done in time $O(n)$.

2. Let $s \in \{0,1\}^n$ be the binary representation of $\lceil p \cdot 2^n \rceil$, i. e., $|s| = \lceil p \cdot 2^n \rceil$. In order to approximate the probability p, we compute $t = \lceil -\log p - \log \varepsilon \rceil$ random bits $c_{n-1}(x), \ldots, c_{n-t}(x)$ with $c_i(x) = IP(x, r^{(i)}) \oplus r_n^{(i)}$ where the $r^{(i)} \in \{0,1\}^n$ and $r_n^{(i)} \in \{0,1\}$ are chosen independently uniformly at random. Now, our random function $f_r(x)$ is equal to 1 iff $|c_{n-1} \cdots c_{n-t} 0^{n-t}| \leq |s|$. In order to construct the OBDD for f_r, we simulate the t OBDDs representing c_i on input $x \in \{0,1\}^n$ in parallel: Since all OBDDs have width 2, we can represent the states of the OBDDs representing $c_{n-1}, \ldots, c_{n-t}$ after reading the same k input variables with at most 2^t OBDD nodes for each $1 \leq k \leq n$. After reading all n input bits, we know the values of $c_{n-1}(x), \ldots, c_{n-t}(x)$ and can easily decide whether $|c_{n-1} \cdots c_{n-t} 0^{n-t}| \leq |s|$ because s is a constant. Therefore, the overall OBDD size is $O(n \cdot 2^t) = O(\frac{n}{p \cdot \varepsilon})$. Each $c_i(x)$ is generated by a BCH scheme for 3-wise independent random bits and $c_i(x)$ and $c_j(x)$ are independent for $i \neq j$. Therefore, $f_r(x)$ is also 3-wise independent.

Let $p' \in (0,1)$ such that $2^n \cdot p'$ is the value of the binary number consisting of the first t most significant bits of s followed by $n-t$ ones, i. e., $2^n \cdot p' = |s_{n-1} \cdots s_{n-t} 1^{n-t}|$. The function f_r ignores the $n-t$ least significant bits of s, therefore, it is equivalent to choose a random vector $v \in \{0,1\}^n$ and check whether $|v| \leq 2^n \cdot p'$ which means that the probability of $f_r(x) = 1$ is p'. It is

$$|s| \leq p' \cdot 2^n \leq |s| + 2^{n-t} - 1 \leq |s| + \varepsilon \cdot p \cdot 2^n \leq (1+\varepsilon)|s|$$

and thus

$$\frac{\lceil p \cdot 2^n \rceil}{2^n} \leq \Pr[f_r(x) = 1] = p' \leq (1+\varepsilon) \cdot \frac{\lceil p \cdot 2^n \rceil}{2^n}.$$

3. For a random matrix $A \in \{0,1\}^n \times \{0,1\}^n$ define $r_i(x) = a_i^T \cdot x = IP(a_i, x)$ where a_i is the i-th column vector of A. Let $f_A(x,i) = 1$ iff $r_i(x) = 1$. An OBDD reading the bits of i first and then computing the corresponding value of $r_i(x)$ has a size of $O(n^2)$ since each $r_i(x)$ can be computed by an OBDD with width 2 (TODO: experiments suggest rather size of $O(n)$). This OBDD can also be constructed in time $O(n^2)$. Let $v_x = Ax$ for $x \in \{0,1\}^n$. Now define $GT_A(x,y) = 1$ iff $|v_x| > |v_y|$, i. e.,

$$GT_A(x,y) = \exists i : f_A(x,i) \wedge \overline{f_A(y,i)} \wedge (\forall j : (j > i) \Rightarrow (f_A(x,i) \Leftrightarrow f_A(y,i))).$$

Since each component of v_x and v_y are pairwise independent, the entire random vectors are also pairwise independent.

The function $g_A(x)$ can be constructed in the same way by replacing $f_A(y,i)$ with $b(i)$ which is equal to the i-th bit of $\lceil p \cdot 2^n \rceil$ and a negation of the resulting function. □

Can we also construct small OBDDs for k-wise independent random variables with $k \geq 4$? Unfortunately, this is not possible.

Theorem 3.2. *Let $X_0, \ldots, X_{2^n-1} : S \to \{0,1\}$ be k-wise independent 0/1-random variables over a sample space S with $\Pr[X_j = 1] = p$ for all $0 \leq j \leq 2^n - 1$ and $k \geq 4$. For every $s \in S$ let $f_s : \{0,1\}^n \to \{0,1\}$ be defined by $f_s(x) := X_{|x|}(s)$. Then, for a fixed variable order π, the expected π-OBDD size of f_s is bounded below by $\Omega(2^{n/3} \cdot (p')^{(1/3)})$ with $p' = 2p(1-p)$.*

Algorithm 1 RandomFunc(x,n)

Input: Variable vector x of length $n \in \mathbb{N}$

Output: 3-wise independent function $r(x)$

Let $r_0, \ldots, r_n$ be $n+1$ independent random bits
$f_r(x) = \bigoplus_{i=0}^{n-1}(r_i \wedge x_i) \oplus r_n$
return $f_r(x)$

Proof. For the sake of simplicity we omit the index of the function f_r. W.l.o.g. let π be the identity order, i.e. $\pi(i) = i$ for all $i = 0, \ldots, n-1$. For $l \in \{1, \ldots, n\}$ and $\alpha \in \{0,1\}^l$ let $f_\alpha : \{0,1\}^{n-l} \to \{0,1\}$ be the subfunction of f where the first l variables $x_0, \ldots, x_{l-1}$ are fixed according to α, i.e. $f_\alpha(z) := f_{|x_0=\alpha_0,\ldots,x_{l-1}=\alpha_{l-1}}(z)$. Now we fix two different assignments α, α' and define 2^{n-l} random variables $D(z) := D_{\alpha,\alpha'}(z)$ such that $D(z) = 1$ iff $f_\alpha(z) \neq f_{\alpha'}(z)$. Since the function values of f are also k-wise independent, for every $z \in \{0,1\}^{n-l}$ we have $E[D(z)] = 2p(1-p) := p'$ and $Var[D(z)] = E[D(z)^2] - E[D(z)]^2 = E[D(z)] - E[D(z)]^2 = p'(1-p')$. Let $D = \sum_z D(z)$. We want to find an upper bound on the number of pairs (α, α') with $f_\alpha = f_{\alpha'}$. The probability that for fixed (α, α') the subfunctions are equal is bounded above by the probability that the difference between D and $E[D]$ is at least $E[D]$, i.e. $\Pr[f_\alpha = f_{\alpha'}] = \Pr[D = 0] \leq \Pr[|D - E[D]| \geq E[D]]$. Each random variable $D(z)$ depends on two function values, i.e. these variables are $k' = \lfloor k/2 \rfloor$-wise independent. Since $k' \geq 2$ we can use Chebyshev´s inequality

$$\Pr[f_\alpha = f_{\alpha'}] \leq \frac{Var[D]}{E[D]^2} = \frac{\sum_z Var[D(z)]}{(2^{n-l} \cdot p')^2} = \frac{2^{n-l} \cdot p' \cdot (1-p')}{(2^{n-l} \cdot p')^2} \leq \frac{1}{2^{n-l} \cdot p'}$$

Hence, the expected number of pairs (α, α') with $f_\alpha = f'_\alpha$ is bounded above by $\frac{\binom{2^l}{2}}{2^{n-l} \cdot p'} \leq \frac{2^{2l}}{2^{n-l} \cdot p'}$. Therefore, the expected number t_l of functions which are equal can be bounded above by $\sqrt{\frac{2^{2l}}{2^{n-l} \cdot p'}} = \frac{2^l}{\sqrt{2^{n-l} \cdot p'}}$. The number s_l of different subfunctions f_α is bounded below by 2^l divided by an upper bound on the number t_l of subfunctions f_α which are equal, i. e., $E[s_l] \geq E\left[\frac{2^l}{t_l}\right] \geq \frac{2^l}{E[t_l]}$ where the last inequality is due to Jensen's inequality and the fact that $g(x) = x^{-1}$ is convex on $(0, \infty)$. The expected number of equal subfunctions can be lower than 1, therefore we have to do a case study:

1. $\frac{2^l}{\sqrt{2^{n-l} \cdot p'}} \leq 1 \Leftrightarrow l \leq (1/3)(n + \log(p'))$: All 2^l subfunctions are different (in expectation).

2. $\frac{2^l}{\sqrt{2^{n-l} \cdot p'}} > 1$: The number of different subfunctions is at least $\frac{2^l \cdot \sqrt{2^{n-l} \cdot p'}}{2^l} = \sqrt{2^{n-l} \cdot p'}$.

For the sake of simplicity, we assume that $(1/3)(n+\log(p'))$ is an integer, since this does not affect the asymptotic behavior. Due to the first case, we know that the number of different subfunctions has to double after each input bit on the first $(n/3)+\log(p')/3+1$ levels, i. e., each node must have two outgoing edges to two different nodes which also means that the all subfunctions essentially depend on the next variable. Therefore, the π-OBDD has to be a complete binary tree on the first $(n/3)+\log(p')/3+1$ levels and the expected π-OBDD size is also $\Omega(2^{n/3}\cdot(p')^{1/3})$.

□

The following theorem shows that k-wise independent random functions with $k \geq 4$ are hard even for FBDDs (and with it for OBDDs and all variable orders). The general strategy of the proof of the next theorem is similar to the proof in [40] where the OBDD size of completely independent random functions was analyzed: We bound the probability p_l that there is a variable order such that the number of OBDD nodes on level l deviates too much from the expected value. If $\sum_{l=0}^{n-1} p_l < 1$ holds, then with probability $1-\sum_{l=0}^{n-1} p_l > 0$ there is no such deviation in any level of the OBDD for all variable orders. The differences lie in the detail: In [40] the function values are completely independent and, therefore, the calculation can be done more directly and with better estimations. We have to take the detour over the number of subfunctions which are equal (as in Theorem 3.2) and can use only Markov's inequality to calculate the deviation of the expectation. Furthermore, because of the independence Wegener [40] was able to do a more subtle analysis of the OBDD size by investigating the effects of the OBDD minimization rules separately.

Theorem 3.3. *Let $X_0,\ldots,X_{2^n-1}: S \to \{0,1\}$ be k-wise independent $0/1$-random variables over a sample space S with $\Pr[X_j=1]=p$ for all $0 \leq j \leq 2^n-1$ and $k \geq 4$. For every $s \in S$ let $f_s: \{0,1\}^n \to \{0,1\}$ be defined by $f_s(x) := X_{|x|}(s)$. Then, there is an r-mixed function f_s with $r=\Omega(n+\log(p')-\log n)$ with $p'=2p(1-p)$.*

Proof. First, we bound the probability that the number t_l of subfunctions which are equal deviates by a factor of δ_l from the expectation. Second, as in Theorem 3.2, we show an upper bound on the level l for which the number of equal subfunctions is lower or equal than 1, i. e., the OBDD has to be a complete binary tree until this level.

As we know, the expected number of pairs (α,α') with $f_\alpha = f'_\alpha$ is bounded above by $\mu_l := \frac{2^{2l}}{2^{n-l}\cdot p'}$. Due to the dependencies, using Markov's inequality is the best we can do to bound the deviation from the expectation. Thus, we have

$$\Pr[\text{No. pairs } (\alpha,\alpha') \text{ with } f_\alpha = f'_\alpha \geq \delta_l \cdot \mu_l] \leq \frac{1}{\delta_l}.$$

Due to Theorem 2.3, the definition of the subfunctions corresponding to OBDD nodes on level l, i. e., the definition of the subfunctions f_α, depends only on the first l variables with respect to the variable order. Thus, we have to distinguish only $\binom{n}{l}$ possibilities to choose these variables. Let $\delta_l := \binom{n}{l}\cdot(n+1)$. Then the probability, that for all levels and variable orders the number of pairs (α,α') with $f_\alpha = f_{\alpha'}$ is at most $\delta_l\cdot\mu_l$ is bounded

below by $1 - n/(n+1) > 0$. Note that this also implies that $t_l \leq \sqrt{\delta_l \cdot \mu_l}$ for all levels l and variable orders.

As in the proof of Theorem 3.2, the next step consists of the investigation of two cases: $\sqrt{\delta_l \mu_l} \leq 1$ and $\sqrt{\delta_l \mu_l} > 1$. Here, we focus only on the first case. In other words, we compute an upper bound T such that $\sqrt{\delta_l \mu_l} \leq 1$ or, equivalently, $(1/2)\log(\delta_l \mu_l) \leq 0$ for all $l \leq T$. For the calculations, we need a known bound for the binomial coefficient $\log \binom{n}{k} \leq n \cdot H(k/n)$ where $H(x) = -x\log(x) - (1-x)\log(1-x)$ is the binary entropy function. It holds

$$\frac{1}{2}\log(\delta_l \mu_l) \quad \leq \quad \frac{1}{2}(3l - n + \log(n) + 1 - \log(p') + n \cdot H\left(\frac{l}{n}\right)).$$

Let $l = \varepsilon \cdot n$ for some $\varepsilon < 1/2$. We want to maximize ε such that $\log(\sqrt{\delta_l \mu_l}) \leq 0$.

$$\begin{aligned} & \frac{1}{2}(3(\varepsilon n) - n + \log(n) + 1 - \log(p') + n \cdot H(\varepsilon)) \quad \leq \quad 0 \\ \Leftrightarrow \qquad & 3\varepsilon + H(\varepsilon) \quad \leq \quad 1 - \frac{\log(1/p')}{n} - \frac{1}{n} - \frac{\log n}{n} \end{aligned}$$

Using $\dfrac{1}{1-x} \leq 1 + 2x$ for $0 \leq x \leq 1/2$ and $\log(1+x) \leq x$ for $x > -1$, we can bound $3\varepsilon + H(\varepsilon)$ by $6\sqrt{\varepsilon}$ (see Appendix for the details). Thus, if

$$\varepsilon \leq \sqrt{\varepsilon} \leq \frac{1}{6} - \frac{1}{6} \cdot \left(\frac{\log(1/p')}{n} + \frac{1}{n} + \frac{\log n}{n} \right) = \Omega\left(1 - \frac{\log(1/p') + \log n}{n}\right),$$

it is $\log(\sqrt{\delta_l \mu_l}) \leq 0$. Since $l = \varepsilon \cdot n$ and the maximal ε is in $\Omega\left(1 - \frac{\log(1/p') + \log n}{n}\right)$ such that $\log(\sqrt{\delta_l \mu_l}) \leq 0$, there is a function f_s which is r-mixed with $r = \Omega(n + \log(p') - \log n)$. □

Due to Lemma 2.5, the last Theorem gives us an lower bound even for FBDDs.

Corollary 3.4. *Let $X_0, \ldots, X_{2^n-1}$ be k-wise independent 0/1-random variables over a sample space S with $\Pr[X_j = 1] = p$ for all $0 \leq j \leq 2^n - 1$ and $k \geq 4$. For every $s \in S$ let $f_s : \{0,1\}^n \to \{0,1\}$ be defined by $f_s(x) := X_{|x|}(s)$. Then, there is a function f_s such that the FBDD size is at least $2^{\Omega(n + \log(p') - \log n)}$.*

4 Construction of Almost k-wise Independent Random Functions.

The gap between the OBDD size of 3-wise independent random functions and 4-wise independent random functions is exponentially large. In order to see what kind of random functions have an OBDD size which is in between these bounds, we show that a construction of a random OBDD due to [11] of size $O((nk)^2/\varepsilon)$ generates (ε, k)-wise independent functions. The idea is to construct a random OBDD with fixed width w. If w is large enough, the function values of k different inputs are almost uniformly

distributed because the paths of the k inputs in the OBDD are likely to be almost independent. For $0 \leq i \leq n-1$ let layer L_i consists of w nodes labeled by x_i and layer L_n be the two sinks. For all $0 \leq i \leq n-1$ we choose the 0/1-successors of every node in layer L_i independently and uniformly at random from the nodes in layer L_{i+1}. Then we pick a random node in layer L_0 as the root of the OBDD.

Theorem 4.1. *For $w \geq k + nk(k+1)/\varepsilon$ the above random process generates (ε, k)-wise independent random functions.*

Proof. Let $a_1, \ldots, a_k \in \{0,1\}^n$ be k different inputs and p be the probability that the function values of these inputs are $\alpha_1, \ldots, \alpha_k \in \{0,1\}$. Let $P_1, \ldots, P_k$ the k paths of $a_1, \ldots, a_k$ to the layer L_{n-1}, i. e., the paths end in a node labeled by x_{n-1}. Let D_i be the event that the paths $P_1, \ldots, P_i$ end in different nodes. Since the inputs are different, every P_i has to use an edge which is not used by any other path and, therefore, it holds $\Pr[D_i \mid D_{i-1}] \geq (1 - \frac{i-1}{w})^n$ and with it $\Pr[D_k] = \prod_{i=2}^{k} \Pr[D_i \mid D_{i-1}] \geq \prod_{i=2}^{k} (1 - \frac{i-1}{w})^n$. We have $\prod_{i=2}^{k} (1 - \frac{i-1}{w})^n \geq \prod_{i=2}^{k} e^{-\frac{n}{w/i-1}} \geq 1 - \varepsilon$ for $w \geq k + nk(k+1)/\varepsilon \geq k + nk(k+1)(1/\ln(\frac{1}{1-\varepsilon}))$. If all paths end in different nodes, then the function values of the k inputs are independent and uniformly distributed, i. e., $p \geq 2^{-k} \cdot \Pr[D_k] \geq 2^{-k} - \varepsilon$ and $p \leq 1 - (1 - 2^{-k}) \cdot \Pr[D_k] \leq 2^{-k} + \varepsilon$ which completes the proof. □

5 Randomized Implicit Algorithms

Complexity Class

Only a small modification is necessary to extend Sawitzki's simulation results from [36] and [38] to show that the set of problems which can solved by a randomized implicit algorithm is equal to the set of problems solved by a randomized parallel algorithm. In the implicit setting, we just add the possibility to construct random functions $r : \{0,1\}^l \to \{0,1\}$ with $l = O(\log N)$. Constructing such functions in parallel is easy. The other way round, i. e., simulating a randomized parallel algorithm by a randomized implicit algorithm, the only difference is the set of input variables of the circuit (which represents the (randomized) parallel algorithm). A deterministic circuit has only N input variables whereas the random circuit has additional $O(N^c)$ random inputs for a constant c. Assuming we can construct a random function $r : \{0,1\}^l \to \{0,1\}$ with $l = O(\log N)$, we can set the input variables correctly for the simulation (in the same way as in [38]).

Randomized Maximal Matching Algorithm

We use the construction of 3-wise independent random functions from the last section to design a randomized maximal matching algorithm. Here, the main drawback of our random construction is the missing possibility to use different probabilities for the nodes.

Algorithm 2 Randomized implicit maximal matching algorithm

Input: Graph $\chi_E(x, y)$
Output: Maximal matching $\chi_M(x, y)$

$\chi_M(x, y) = 0$ // Initial matching
while $\chi_E(x, y) \not\equiv 0$ **do**
  $\chi_{E'}(x, y) = \chi_E(x, y)$
  // Compute set of nodes with two or more incident edges
  $T(x) = \exists z, y : (z \neq y) \wedge \chi_{E'}(x, y) \wedge \chi_{E'}(x, z)$
  $NewEdges(x, y) = 0$
  while $T(x) \not\equiv 0$ **do**
    // Construct 3-wise independent random functions (see Algorithm 1)
    $f_{r_1}(x) = RandomFunc(x, n)$ and $f_{r_2}(y) = RandomFunc(y, n)$
    $F(x, y) = (x > y) \wedge (f_{r_1}(x) \oplus f_{r_2}(y))$
    $F(x, y) = F(x, y) \vee F(y, x)$
    $\chi_{E'}(x, y) = \chi_{E'}(x, y) \wedge F(x, y)$ // Delete edges with probability 1/2
    $T(x) = \exists z, y : (z \neq y) \wedge \chi_{E'}(x, y) \wedge \chi_{E'}(x, z)$ // Update $T(x, y)$
    // Store isolated edges in NewEdges
    $NewEdges(x, y) = NewEdges(x, y) \vee (\chi_{E'}(x, y) \wedge \overline{T(x)} \wedge \overline{T(y)})$
  end while
  $\chi_M(x, y) = \chi_M(x, y) \vee NewEdges(x, y)$ // Add edges to current matching
  $Matched(x) = \exists y : \chi_M(x, y)$
  $\chi_E(x, y) = \chi_E(x, y) \wedge \overline{Matched(x)} \wedge \overline{Matched(y)}$ // Delete edges incident to matched nodes
end while
return $\chi_M(x, y)$

Randomized algorithms for maximal independent set using pairwise independence like in [1] or [27] choose a node with a probability proportional to the node degree. In order to simulate these selections by our construction, we delete each edge with probability 1/2 as long as there are other incident edges. Finally, we add the remaining isolated edges to the matching. Algorithm 2 shows the whole randomized implicit maximal matching algorithm. We realize the edge deletions of the inner loop in the following way: We construct two 3-wise independent random functions $f_{r_1}(x), f_{r_2}(y)$ using Algorithm 1 and set $F(x, y) = (x > y) \wedge (f_{r_1}(x) \oplus f_{r_2}(y))$. Since $\Pr_{r_1,r_2}[f_{r_1}(x) \oplus f_{r_2}(y) = 1] = \Pr_{r_1,r_2}[f_{r_1}(x) \neq f_{r_2}(y)] = 1/4+1/4 = 1/2$ for inputs $x \leq y$ the function $F(x, y)$ deletes such edges as required. Because we are dealing with undirected graphs, we want $F(x, y) = F(y, x)$ for every (x, y). Therefore, we set $F(x, y) = F(x, y) \vee F(y, x)$ and delete the edges with the operation $\chi_E(x, y) = \chi_E(x, y) \wedge F(x, y)$.

We say that an edge $e \in E'$ (before the inner while-loop) survives iff $e \in E'$ after the inner while-loop of algorithm 2.

Lemma 5.1. *For every $e = \{u, v\} \in E$ with $deg_E(u) > 1$ or $\deg_E(v) > 1$ before the inner while-loop in algorithm 2 the probability that e survives is at least $\frac{1}{8 \cdot (deg_E(u) + \deg_E(v) - 2)}$.*

Proof. Let $e = \{u, v\} \in E$ be an edge before the inner while-loop and R_e be the number of rounds until edge e is deleted. The random bits in each iteration are 3-wise independent and the iterations themselves are completely independent. Thus, the variables R_e are also 3-wise independent. Denote by $N(e) = \{e' \in E \mid e \cap e' \neq \emptyset\}$ the neighborhood of e, i. e., all edges incident to u or v. Then we have $\Pr[e \text{ survives}] = \Pr[R_e \text{ is unique maximum in } \{R_{e'} \mid e' \in N(e)\}]$. It is easy to see that $\Pr[R_e = i] = \left(\frac{1}{2}\right)^i$ for $i \geq 1$. Let $e' \in N(e)$ and $e' \neq e$ and $z \geq 1$ be fixed. Since the R_e are 3-wise independent, we have $\Pr[R_{e'} \geq z \mid R_e = z] = \Pr[R_{e'} \geq z] = \sum_{i=z}^{\infty} \left(\frac{1}{2}\right)^i = \left(\frac{1}{2}\right)^{z-1}$. Therefore, the probability that there is an edge $e' \in N(e) \setminus e$ with $R_{e'} \geq z$ is at most $\frac{|N(e)|-1}{2^{z-1}}$, i. e., R_e is unique maximum with probability at least $1 - \frac{|N(e)|-1}{2^{z-1}}$. This is greater than 0 for $z \geq \log(|N(e)| - 1) + 2$. Finally, we have $\Pr[R_e \text{ is unique maximum}] \geq \left(\frac{1}{2}\right)^{\log(|N(e)|-1)+2} \cdot \left(1 - \frac{|N(e)|-1}{2^{\log(|N(e)|-1)+1}}\right) \geq \frac{1}{8 \cdot (deg_E(u) + \deg_E(v) - 2)}$ □

The number of deleted edges for a matching edge (u, v) that is added to the matching is $deg(u) + deg(v) - 2$ if we do not count the matching edge itself. Thus, the expected number of deleted edges is $\Omega(|E|)$ at the end of the outer loop. This gives us the final result.

Theorem 5.2. *Let $G = (V, E)$ be a graph with N nodes. All functions used in algorithm 2 on the input χ_E depend on at most $3 \log N$ variables. The expected number of operations is $O(\log^3 N)$.*

Proof. Each iteration of the inner-loop needs $O(\log N)$ operations. Since we halve the number of edges in expectation in each iteration of this loop, the expected number of iterations is $O(\log N)$. The edges surviving the inner loop are those that are added to the matching. After adding a set of edges to the matching, all edges that are incident to a matched node are deleted from the graph in the outer loop. The number deleted edges for a matching edge (u, v) that is added to the matching is $deg(u) + deg(v) - 2$ if we do not count the matching edge itself. Thus, by Lemma 3, the expected number of edges deleted in this step is at least

$$\sum_{e=\{u,v\} \in E} (deg_E(u) + deg_E(v) - 2) \cdot \frac{1}{8 \cdot (deg_E(u) + \deg_E(v) - 2)} = |E|/8.$$

This implies that the expected number of iterations of the outer-loop is also bounded above by $O(\log N)$. □

Application to the Maximal Independent Set Problem

With a similar idea we are able to design a distributed MIS algorithm: Each node v draws a random bit until this bit is 0. Let r_v be the number of bits drawn by node v.

We send r_v to all neighbors and include node v to the independent set iff r_v is a local minimum. The expected number of bits for each channel is 1. A similar analysis as before show that we have an maximal independent set after $O(\log N)$ steps in expectation and the overall expected number of bits per channel is $O(\log N)$.

Experimental Results.

All algorithms are implemented in C++ using the BDD framework CUDD 2.5.0[1] by F. Somenzi and were compiled with Visual Studio 2013 in the default 32-bit release configuration. All source files, scripts and random seeds will be publicly available[2]. The experiments were performed on a computer with a 2.5 GHz Intel Core i7 processor and 8 GB main memory running Windows 8.1. The runtime is measured by used processor time in seconds and the space usage of the implicit algorithm is given by the maximum SBDD size which came up during the computation, where an SBDD is a collection of OBDDs which can share nodes. Note that the maximum SBDD size is independent of the used computer system. For our results, we took the mean value over 50 runs on the same graph. Due to the small variance of these values, we only show the mean in the diagrams/tables. We omit the algorithm by Bollig and Pröger [10] because the memory limitation was exceeded on every instance presented here.

We choose three types of input instances: First, we used our construction from section 4 as an input distribution in the following way: If the 1 sink is chosen with probability p as a successor of nodes in layer L_{n-1} the expected size of $|f^{-1}(x)|$ is $p \cdot 2^n$. For a fixed $N = 2^{17}$, we used p as a density parameter for our input graph and want to analyze how the density influences the running time of the algorithms. Second, we run the algorithms on some bipartite graphs from a real advertisement application within Google[3] [32]. The motivation was to check whether the randomized algorithm is competitive or even better on instances where the algorithm by Hachtel and Somenzi (HS) [20] is running very well. Third, we use non-bipartite graphs from the university of Florida sparse matrix collection [16]. Since HS is designed for bipartite graphs, a preprocessing step computing a bipartition of these graphs are needed to compute a maximal matching (see, e. g., [10]) while our algorithm also works on general graphs.

In the experiments we use the following implementation of our algorithm denoted by RM. In order to minimize the running time for computation of the set of nodes with two or more incident edges, we sparsify the graph at the beginning of the outer while loop by deleting each edge with probability 1/2 and repeating this D times. Initially, we set $D = \log |E|$ and decrease D by 1 at the end of the outer loop. Asymptotically, the running time does not change since after $O(\log N)$ iterations, i. e., $D = 0$, it does exactly the same as original algorithm. Initial experiments showed that this is superior to the original algorithm.

[1] `http://vlsi.colorado.edu/~fabio/CUDD/`

[2] `http://ls2-www.cs.uni-dortmund.de/~gille/`

[3] Graph data files can be found at `http://www.columbia.edu/~cs2035/bpdata/`

Instance	Nodes	Edges	Time (sec)	Space (SBDD size)
333SP	3712815	22217266	1140.54	66968594
adaptive	6815744	27248640	403.82	22767094
as-Skitter	1696415	22190596	337.53	32020282
hollywood-2009	1139905	113891327	418.36	62253086
roadNet-CA	1971281	5533214	136.18	13177668
roadNet-PA	1090920	3083796	75.26	7633318
roadNet-TX	1393383	3843320	92.62	9125438

Table 1: Running time and space usage of RM on the graphs from [16]

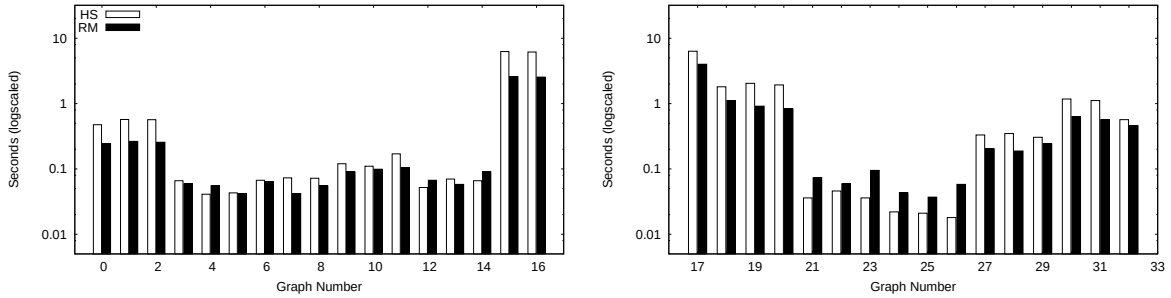

Figure 2: Running times of HS and RM on the real world instances.

On the random instances the running time and space usage of RM was more or less unaffected by the density of the graph while HS was very slow for small values of p and gets faster with increasing density. For $p \leq 0.2$ RM was much faster than HS (see Fig. 3 4). In Fig. 2 we see that on the bipartite real world instances RM is similar to HS if the running time is negligibly small but on the largest instances (number 15 to 20) RM is much faster. the graphs from [16] were intentionally chosen to show the potential of RM and indeed do so: It was not possible to run HS on these graphs due to memory limitations whereas RM computed a matching in reasonable time and space (see Table 1). Both graphs from and [16] have very small density and the experiments on the random graphs seem to support the hypothesis that RM is a better choice than HS for such graphs.

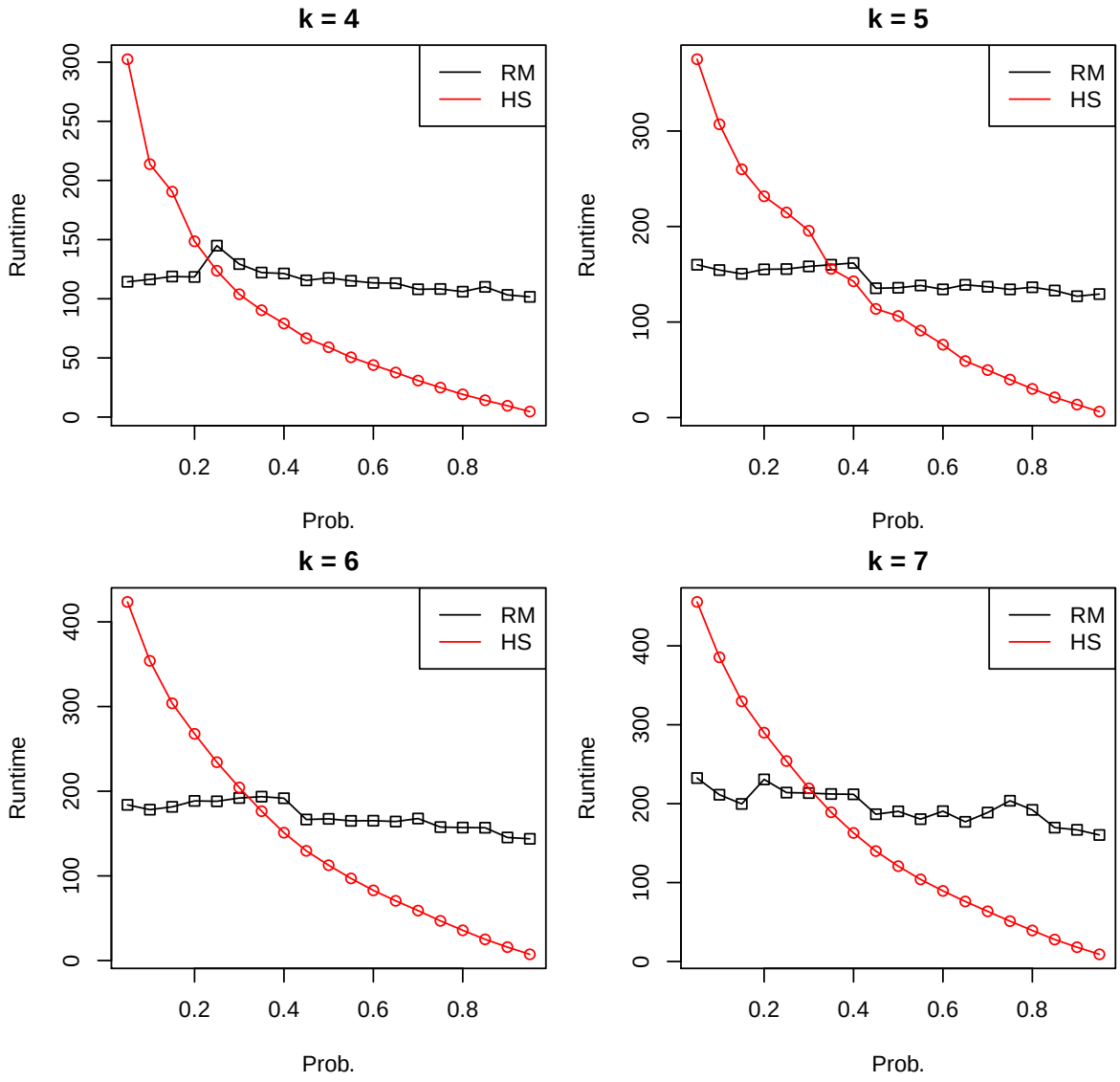

Figure 3: Running times of HS and RM on the random instances.

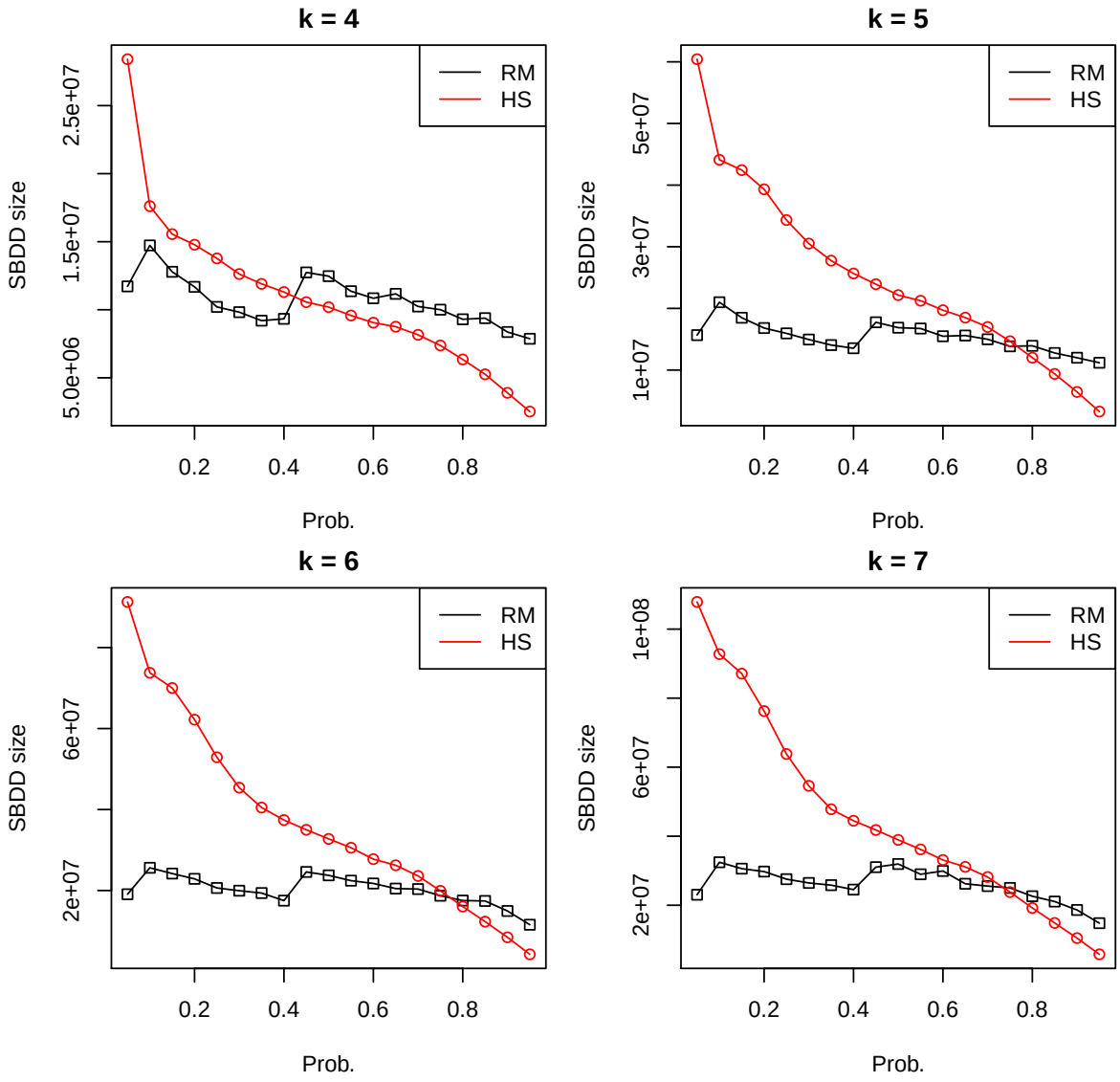

Figure 4: Space usage of HS and RM on the random instances.

Further Applications.

Extending our matching algorithm to the more general f-matching, where each node v is allowed to have at most $f(v)$ incident matching edges, is an interesting question. Designing other randomized implicit algorithms, e. g., for minimum spanning tree, where random sampling of subgraphs are necessary, seems straightforward and initial experiments showed that this could lead to faster algorithms than the known deterministic ones.

Acknowledgements

I would like to thank Beate Bollig, Melanie Schmidt and Chris Schwiegelshohn for the valuable discussions and for their comments on the presentation of the paper.

References

[1] Alon, N., Babai, L., and Itai, A. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms 7*, 4 (1986), 567 – 583.

[2] Alon, N., Goldreich, O., Håstad, J., and Peralta, R. Simple construction of almost k-wise independent random variables. *Random Struct. Alg. 3*, 3 (1992), 289–304.

[3] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *J. Comp. and System Sc. 58*, 1 (1999), 137 – 147.

[4] Awerbuch, B., Goldberg, A. V., Luby, M., and Plotkin, S. A. Network decomposition and locality in distributed computation. In *FOCS* (1989), pp. 364–369.

[5] Bloem, R., Gabow, H. N., and Somenzi, F. An algorithm for strongly connected component analysis in $n\log n$ symbolic steps. *Formal Meth. in System Design 28*, 1 (2006), 37–56.

[6] Bollig, B. On symbolic OBDD-based algorithms for the minimum spanning tree problem. *Theor. Comput. Sci. 447* (2012), 2–12.

[7] Bollig, B., and Capelle, M. Priority functions for the approximation of the metric TSP. *Inf. Proc. Letters 113*, 14-16 (2013), 584–591.

[8] Bollig, B., Gillé, M., and Pröger, T. Implicit computation of maximum bipartite matchings by sublinear functional operations. In *TAMC* (2012), pp. 473–486.

[9] Bollig, B., Löbbing, M., and Wegener, I. On the effect of local changes in the variable ordering of ordered decision diagrams. *Inf. Proc. Letters 59*, 5 (1996), 233–239.

[10] Bollig, B., and Pröger, T. An efficient implicit OBDD-based algorithm for maximal matchings. In *LATA* (2012), pp. 143–154.

[11] Bollig, B., and Wegener, I. personal communication, 2014.

[12] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers 35*, 8 (1986), 677–691.

[13] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. Symbolic model checking: 10^{20} states and beyond. *Inf. and Comp. 98*, 2 (1992), 142–170.

[14] Chor, B., and Goldreich, O. On the power of two-point based sampling. *J. Complexity 5*, 1 (1989), 96–106.

[15] Coudert, O. Doing two-level logic minimization 100 times faster. In *SODA* (1995), pp. 112–121.

[16] Davis, T. A., and Hu, Y. The University of Florida Sparse Matrix Collection. *ACM Trans. on Math. Soft. 38*, 1 (Nov. 2011), 1:1–1:25.

[17] Gentilini, R., Piazza, C., and Policriti, A. Computing strongly connected components in a linear number of symbolic steps. In *SODA* (2003), pp. 573–582.

[18] Gentilini, R., Piazza, C., and Policriti, A. Symbolic graphs: Linear solutions to connectivity related problems. *Algorithmica 50*, 1 (2008), 120–158.

[19] Gillé, M. OBDD-based representation of interval graphs. In *WG*, vol. 8165 of *LNCS*. Springer Berlin Heidelberg, 2013, pp. 286–297.

[20] Hachtel, G. D., and Somenzi, F. A symbolic algorithms for maximum flow in 0-1 networks. *F. Meth. in Sys. Design 10*, 2/3 (1997), 207–219.

[21] Hojati, R., Touati, H., Kurshan, R. P., and Brayton, R. K. Efficient ω-regular language containment. In *Comp. Aided Verification*, vol. 663 of *LNCS*. Springer, 1993, pp. 396–409.

[22] Israeli, A., and Itai, A. A fast and simple randomized parallel algorithm for maximal matching. *Inf. Process. Lett. 22*, 2 (1986), 77–80.

[23] Jukna, S. Entropy of contact circuits and lower bounds on their complexity. *Theor. Comput. Sci. 57* (1988), 113–129.

[24] Kabanets, V. Almost k-wise independence and hard boolean functions. *Theor. Comput. Sci. 297*, 1-3 (2003), 281–295.

[25] Lai, Y., Pedram, M., and Vrudhula, S. B. K. EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Trans. on CAD of Int. Circuits and Systems 13*, 8 (1994), 959–975.

[26] Linial, N. Locality in distributed graph algorithms. *SIAM J. Comput. 21*, 1 (1992), 193–201.

[27] Luby, M. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing 15*, 4 (1986), 1036–1053.

[28] Masek, W. A fast algorithm for the string editing problem and decision graph complexity. Master's thesis, MIT, 1976.

[29] Meer, K., and Rautenbach, D. On the OBDD size for graphs of bounded tree- and clique-width. *Discrete Mathematics 309*, 4 (2009), 843–851.

[30] Métivier, Y., Robson, J. M., Saheb-Djahromi, N., and Zemmari, A. An optimal bit complexity randomized distributed MIS algorithm. *Distributed Computing 23*, 5-6 (2011), 331–340.

[31] Naor, J., and Naor, M. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput. 22*, 4 (1993), 838–856.

[32] Negruseri, C. S., Pasoi, M. B., Stanley, B., Stein, C., and Strat, C. G. Solving maximum flow problems on real world bipartite graphs. In *ALENEX* (2009), pp. 14–28.

[33] Nunkesser, R., and Woelfel, P. Representation of graphs by OBDDs. *Discrete Applied Mathematics 157*, 2 (2009), 247–261.

[34] Savický, P. Improved boolean formulas for the ramsey graphs. *Random Struct. Algorithms 6*, 4 (1995), 407–416.

[35] Sawitzki, D. Implicit flow maximization by iterative squaring. In *SOFSEM* (2004), pp. 301–313.

[36] Sawitzki, D. The complexity of problems on implicitly represented inputs. In *SOFSEM* (2006), pp. 471–482.

[37] Sawitzki, D. Exponential lower bounds on the space complexity of OBDD-based graph algorithms. In *LATIN* (2006), pp. 781–792.

[38] Sawitzki, D. Implicit simulation of FNC algorithms. *Electronic Colloquium on Computational Complexity (ECCC) 14*, 028 (2007).

[39] Sieling, D., and Wegener, I. NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters 3* (1993), 3–12.

[40] Wegener, I. The size of reduced OBDDs and optimal read-once branching programs for almost all boolean functions. *IEEE Trans. on Comp. 43*, 11 (1994), 1262–1269.

[41] Wegener, I. *Branching programs and binary decision diagrams.* SIAM Monographs on Discrete Mathematics and Applications, 2000.

[42] Woelfel, P. Symbolic topological sorting with OBDDs. *J. Disc. Alg. 4* (2006), 51–71.

Proof of Theorem 3.3

Claim .3. *Let* $\varepsilon \leq 1/2$. *Then* $3\varepsilon + H(\varepsilon) \leq 6\sqrt{\varepsilon}$.

Proof. Recall that $H(x) = -x\log(x) - (1-x)\log(1-x)$. Using $\frac{1}{1-x} \leq 1 + 2x$ for $0 \leq x \leq 1/2$ and $\log(1+x) \leq x$ for $x > -1$ we have

$$\begin{aligned} 3\varepsilon + H(\varepsilon) &= 3\varepsilon + \varepsilon\log(1/\varepsilon) + (1-\varepsilon)\cdot\log(1/(1-\varepsilon)) \\ &\leq 3\varepsilon + \varepsilon\sqrt{1/\varepsilon} + \log(1/(1-\varepsilon)) \\ &\leq 3\varepsilon + \sqrt{\varepsilon} + 2\varepsilon \\ &\leq 6\sqrt{\varepsilon}. \end{aligned}$$

□

Experiments

Number	Running Time (RM)	Running Time (HS)	SBDD Size (RM)	SBDD Size (HS)
0	**0.243**	0.475	**567210**	1346996
1	**0.264**	0.571	**555968**	1394008
2	**0.256**	0.567	**553924**	1394008
3	**0.059**	0.066	**153300**	220752
4	0.055	**0.041**	**161476**	194180
5	**0.042**	0.043	**153300**	194180
6	**0.064**	0.067	**196224**	252434
7	**0.042**	0.073	**163520**	279006
8	**0.055**	0.072	**169652**	279006
9	**0.09**	0.12	**240170**	368942
10	**0.099**	0.11	**237104**	368942
11	**0.105**	0.17	**245280**	368942
12	0.067	**0.052**	**236082**	245280
13	**0.058**	0.07	**242214**	310688
14	0.091	**0.066**	**284116**	328062
15	**2.565**	6.259	**3115056**	7887796

Table 2: Running times and space usage of RM and HS on real-world instances from [32].

Number	Running Time (RM)	Running Time (HS)	SBDD Size (RM)	SBDD Size (HS)
16	**2.545**	6.167	**3115056**	7874510
17	**4.002**	6.329	**3115056**	7874510
18	**1.112**	1.81	**2053198**	2320962
19	**0.913**	2.043	**2035824**	2485504
20	**0.828**	1.931	**2035824**	2485504
21	0.073	**0.036**	**182938**	231994
22	0.059	**0.046**	**163520**	240170
23	0.095	**0.036**	**162498**	240170
24	0.043	**0.022**	134904	134904
25	0.037	**0.021**	135926	135926
26	0.058	**0.018**	169652	**135926**
27	**0.203**	0.331	**346458**	731752
28	**0.188**	0.348	**317842**	677586
29	**0.244**	0.305	**319886**	677586
30	**0.632**	1.176	**1314292**	2100210
31	**0.568**	1.114	**1280566**	2104298
32	**0.458**	0.568	**950460**	1410360

Table 3: Running times and space usage of RM and HS on real-world instances from [32].

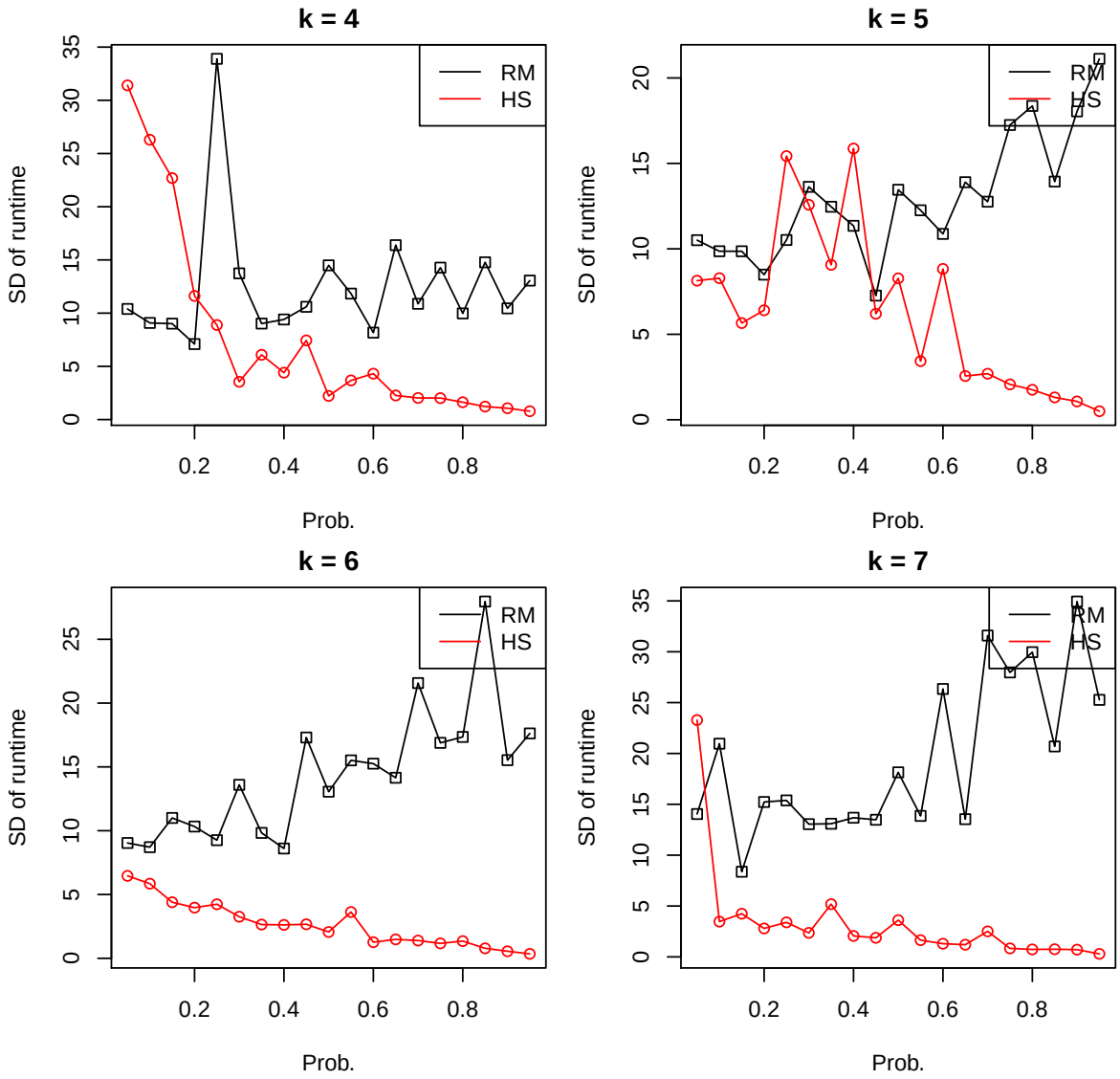

Figure 5: Standard deviations of the running times.

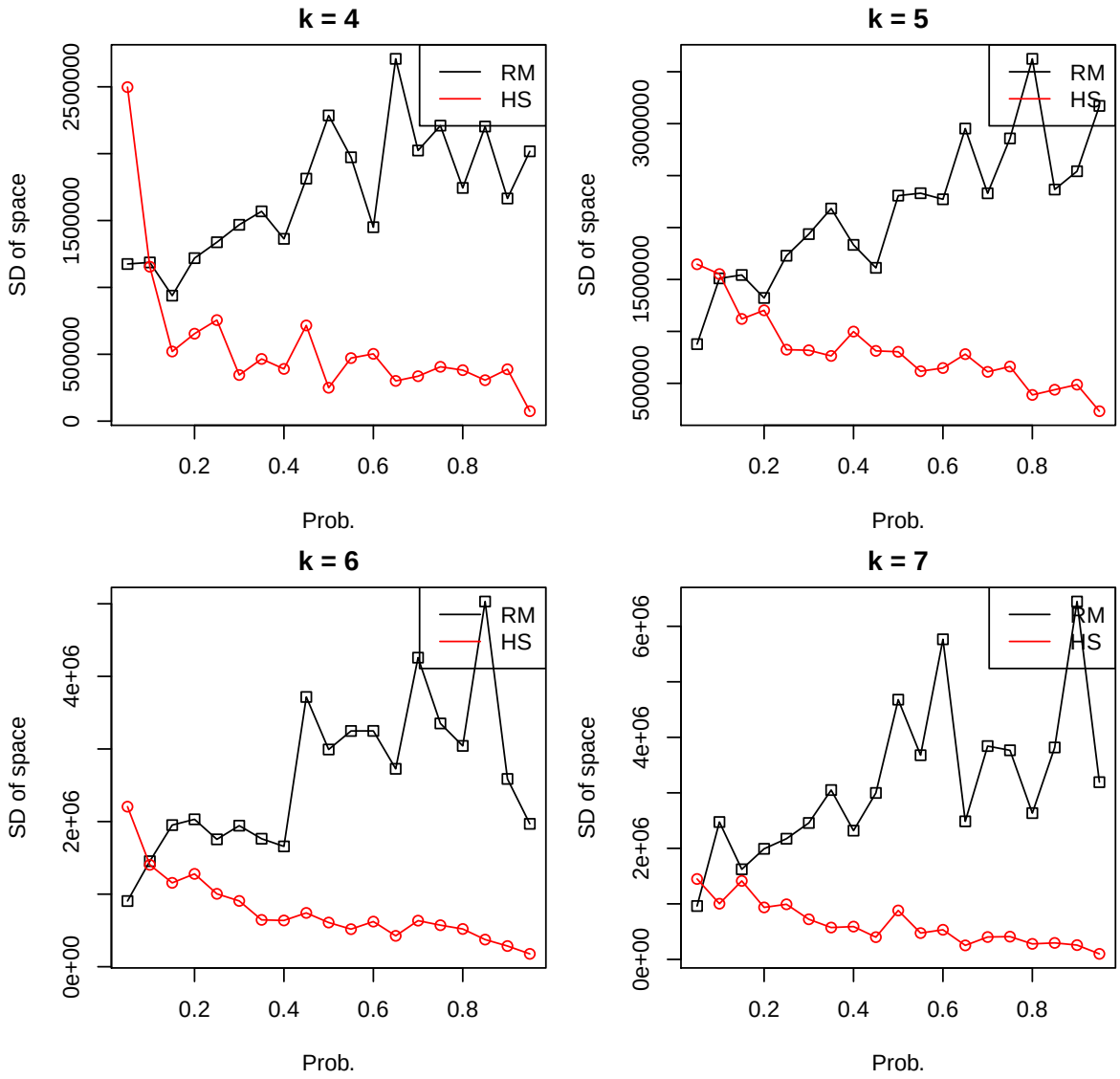

Figure 6: Standard deviation of the space usage.

Number	$\|U\|$	$\|W\|$	Edges
0	136	18872	222951
1	137	18872	222951
2	137	18888	222951
3	40	7086	33609
4	41	7086	33609
5	41	7093	33609
6	125	7107	33609
7	86	7117	33609
8	86	7127	33609
9	289	16653	33051
10	290	16846	33051
11	290	16904	33051
12	50	13360	50040
13	51	16264	56577
14	51	21016	56577
15	164	288826	2523313

Number	$\|U\|$	$\|W\|$	Edges
16	165	288826	2523313
17	165	288858	2523313
18	196	89030	1080027
19	197	89044	1080041
20	197	89044	1080041
21	40	28489	43629
22	41	28944	43629
23	41	28944	43629
24	35	11361	22279
25	36	11588	22279
26	36	11695	22279
27	934	8752	42711
28	935	8896	42711
29	935	9028	42711
30	125	55058	844598
31	126	56858	844598
31	126	57926	477356

Table 4: Properties of the real-world instances from [32].